TP 3: k-Nearest Neighbors (OBLIGATORY)

Thursday 29$^{\text{th}}$ April, 2021
deadline: Friday 14$^{\text{th}}$ May, 2021, 23:59

In this TP, we ask you to implement and apply a k-Nearest Neighbor (kNN) classifier with different distances, such as Euclidean and Mahalanobis. During the implementation of the kNN you will see the importance of vectorizing your code with python. Finally, we ask you to visualize the effect of a transformation in the unit circle.

You are going to fill a few missing functions in the python scripts [1] to implement the exercises that we ask. First of all, read and understand the given python scripts. To run your code, you have to run the jupyter notebook. Here you have **not** to code anything, just to follow it and answer when asked. The code and the notebook given work if the missing functions are correctly implemented!

You are going to use the *CIFAR-10* and the *iris* data set. If you are using **windows** the textitget_dataset.sh script will not work (only mac and linux are available), instead use this link and place the content of the zip inside the dataset folder.

Exercise 1

In this exercise, you are going to see how important it is to vectorize your code using NumPy. To do that, you will compute the Euclidean distance in three different ways. Using two, one, and without for loops and you will run your algorithm for each case using the *CIFAR-10* data set to compare the time performance (using Euclidean distance and $k = 5$).

Exercise 2

Now you will check and comment on how the performance changes with respect to the values of k. How the number of neighbors influence the classification? Run your algorithm using the *iris* data set for $k = [1, 3, 5, 10, 20, 50]$. For each k use the Euclidean and the Mahalanobis distance.

[1] Part of the given code is based on Stanford's repository

The *Euclidean* distance between two learning instances $\mathbf{x}_i \in \mathbf{R}^d$ and $\mathbf{x}_j \in \mathbf{R}^d$, where d is the feature (attribute) dimension, is defined as:

$$d_2(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)}$$

The *Mahalanobis* distance, is parametrized by a $d \times d$ covariance matrix Σ, and is defined as:

$$d(\mathbf{x}_i, \mathbf{x}_j; \Sigma) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T\Sigma^{-1}(\mathbf{x}_i - \mathbf{x}_j)}$$

It is easy to see that the euclidean distance is simply the Mahalanobis distance with the identity matrix I as a covariance matrix.

In the case of the Mahalanobis distance, you are going to explore three different approaches to compute it.

1. Define Σ as a diagonal matrix that has at its diagonal the average variance of the different features, i.e. all diagonal entries Σ_{ii} will be the same ($\Sigma = I\sigma$, where $\sigma = \frac{1}{d}\sum_{k=1}^{d}\sigma_k$).
2. Define Σ as a diagonal matrix that has at its diagonal the variance of each feature, i.e. σ_k.
3. Define Σ as the full covariance matrix between all pairs of features.

- Explain how the performance changes with respect to the values of k and the different distances that you use. How does the number of neighbors influence the classification?
- For a fixed number of neighbors (for example k= 5) comment on the differences between the two distances and on the differences between the three different versions of the Mahalanobis distances. Comment how these affect the performance of the classification, when we should prefer the one over the other, etc..
- ....

Exercise 3

Visualize, study, and discuss the decision surfaces that kNN algorithm produces for the different values of k using the Euclidean distance. To do so, you will work only in two attributes.

- Testing will be done on an artificially generated dataset that covers in a regular manner all possible values for the two chosen attributes. We need to divide the space into a grid by discretizing the space into n values between the minimum and maximum value of an attribute. Each of these values must be compared with the n discrete values of the second attribute. The resulting array will be of shape (n * n, 2)

- Using your training set classify your test instances and visualize the results of the classification

Exercise 4

Linear Transformations:
Draw the unit circle in the two dimensional space, i.e. the set of points for which $\mathbf{x}^T\mathbf{x} = 1$, let us call that graph 1. Graph 1 gives the points that are in distance one from the axes origin. Now given a matrix $\mathbf{A} : 2 \times 2$ show how the points of the unitary circle are transformed when you apply the linear transformation $\mathbf{Ax}$, let us call that graph 2. Draw the unit circle for the transformed space, i.e. the set of points for which $(\mathbf{Ax})^T(\mathbf{Ax}) = 1$, lets us call that graph 3[2]. What is the Mahalanobis matrix associated with the $\mathbf{Ax}$ linear transformation. Comment on the relations of the three graphs and the directions on which distances change the most and the least. Why the direction of the original unit circle that comes closer to the origin after the $\mathbf{Ax}$ transformation is the one that is further away from the center when we consider the $(\mathbf{Ax})^T(\mathbf{Ax}) = 1$ unit circle? Use $\mathbf{A} = \begin{pmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{pmatrix}$.
Draw the unit circles for the three covariance matrices that you consider in exercise 2, i.e. draw the set of points for which $\mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x} = 1$. If you need to compute $\mathbf{A} = \mathbf{\Sigma}^{-1/2}$ use the eigen-value decomposition of $\Sigma = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ ($\mathbf{V}$ has as columns the eigen-vectors of $\mathbf{\Sigma}$ and $\mathbf{\Lambda}$ is a diagonal matrix that contains the eigenvalues of Σ), and define $\mathbf{A}$ as $\mathbf{V}\sqrt{\mathbf{\Lambda}}$.

General instructions

You have to put your work in *cyberlearn* saved in a zip file using as name this format: TP_knn_LASTNAME_Firstname. You can clean the *datasets* folder by running *.clean.sh* before upload your work.

[2] To do so let us define $z = Ax$. For a moment establish the points that belong in unitary circle $z^Tz = 1$, without considering the Ax relation. Now since we know that z=Ax we can solve for x as usually i.e. $A^{-1}z = x$ and this now gives us the set of points in the x representation for which $(Ax)^T(Ax) = 1$.